InHolland

VHDL

S88-n Protocol in VHDL

Authors:

Koen Groot
549543

Ruben Pera
551198

Samenvatting

In dit onderzoeks verslag wordt de werking van het s88-n protocol onderzocht. Deze werking wordt geïmplementeerd met behulp van VHDL.

Inhoudsopgave

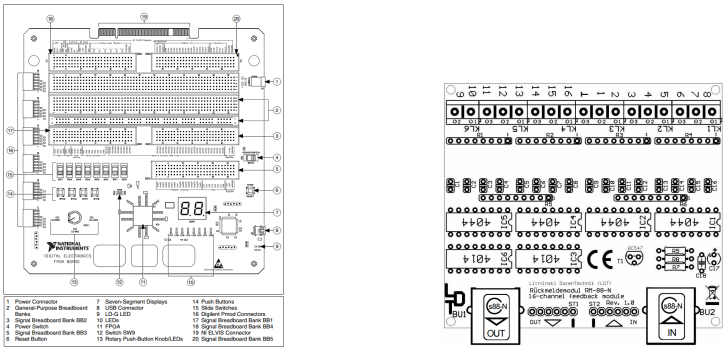
Hoofdstuk 1

Inleiding

Het doel waar naar toe wordt gewerkt in dit onderzoek is om via het S88 timing protocol te communiceren met het RM-88-N schuifregister doormiddel van een NI-FPGA bord te programmeren in de hardware beschrijvingstaal VHDL.

Dit houd in:
- Werkende VHDL code schrijven om de verbinding te leggen met het schuifregister en de betreffende respons op te vangen en weer te geven, zodat er een duidelijk beeld is dat de communicatie tot stand is gebracht.
- Het S88 timing protocol onderzoeken en waarom dit toepasselijk is voor dit onderzoek.

Hier een schematisch beeld van zowel het NI-FPGA bord, als het RM-88-N schuifregister.



Figuur 1.1: Schematische weergaven hardware

Hoofdstuk 2

Probleem Omschrijving

2.1 Probleem Omschrijving

De RM88 is ontworpen om als feedback bus te dienen, de feedback wordt geleverd via het S88 protocol.
De RM88 is een serieel schuif register met een parralelle load input, met het S88 protocol wordt de parralelle input omgezet naar een seriele input.
In dit onderzoek gaat er onderzocht worden of het S88 protocol nagebouwd kan worden met een XILINX Spartan 3E FPGA bord en de hardware omschrijf taal VHDL.

2.2 Hoofdvraag

De Hoofdvraag bij dit onderzoek luidt:

Hoe kan er op een XILINX Spartan 3E FPGA bord met behulp van de hardware beschrijving taal VHDL een s88 protocol geïmplementeerd worden?

2.3 Deelvragen

Om de Hoofdvraag goed te kunnen beantwoorden zijn er de volgende deelvragen opgesteld:

1. Hoe werkt het s88 protocol?
2. Hoe werkt een XILINX Spartan 3E FPGA bord?
3. Hoe kan de hardware beschrijving taal VHDL gebruikt worden op het XILINX bord?
4. Hoe kan een schuifregister aangesloten worden op het XILINX bord?

Hoofdstuk 3

Specificaties

In de specificaties wordt omschreven welke soft- en hardware er gebruikt wordt, met daarbij de relevante karakteristieken, in dit verslag

3.1 Hardware

Er wordt gebruik gemaakt van de volgende Hardware

- Er wordt gebruik gemaakt van een XILINX Spartan 3E FPGA bord.
- Er wordt gebruik gemaakt van een Field Programmable Gate Array (FPGA).

3.2 Software

Er wordt gebruik gemaakt van de volgende Software

- Er wordt gebruik gemaakt van een 64 bit versie van Windows 7.
- Er wordt gebruik gemaakt van Oracle VM VirtualBox om de Windows 7 in te emuleren.
- Er wordt gebruik gemaakt van XILINX Tools voor Windows.

Hoofdstuk 4

Vereisten

In de vereisten wordt omschreven waar het systeem aan moet voldoen bij het afleveren van het systeem.

- Het XILINX Spartan 3E FPGA bord moet kunnen communiceren met een Field Programmable Gate Array (FPGA).
- De communicatie tussen het XILINX bord en de FPGA moet plaatsvinden door middel van het S88 protocol.
- De communicatie tussen het XILINX bord en de FPGA wordt als volgt beschreven, het XILINX bord zal via het S88 protocol de eerste byte uit het FPGA lezen.
- Het XILINX bord zal geconfigureerd worden met de hardware beschrijving taal VHDL.

Hoofdstuk 5

Analyse

De Hoofdvraag luid: ẗo”In dit onderzoek gaat er onderzocht worden of het S88 protocol nagebouwd kan worden met een XILINX Spartan 3E FPGA bord en de harware omschrijf taal VHDL.

5.1 deelvragen

Voor de Analyse zijn er meerdere deelvragen opgesteld,

1. Hoe werkt het s88 protocol?
2. Hoe werkt een XILINX Spartan 3E FPGA bord?
3. Hoe kan de hardware beschrijving taal VHDL gebruikt worden op het XILINX bord?
4. Hoe kan een schuifregister aangesloten worden op het XILINX bord?

5.1.1 Hoe werkt het s88 protocol?

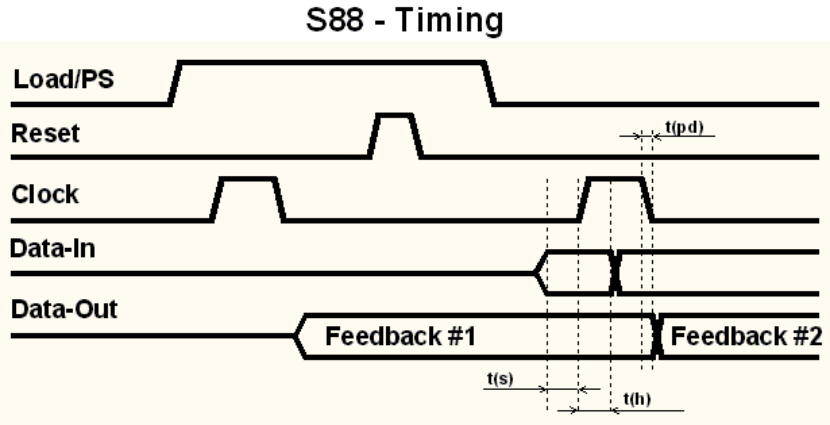
Het s88 wordt uitgelegd met behulp van de twee onderstaande afbeeldingen.

Het Xilinx bord kan met behulp van het s88 protocol de rechter bus uitlezen. Dit gebeurt met behulp van een schuifregister.

Met de DATA OUT poort van het Xilinx bord wordt data ingelezen. Deze is verbonden met de Q1 Out poort van het schuifregister, hiermee wordt dus informatie van het schuifregister naar het Xilinx bord overgedragen.

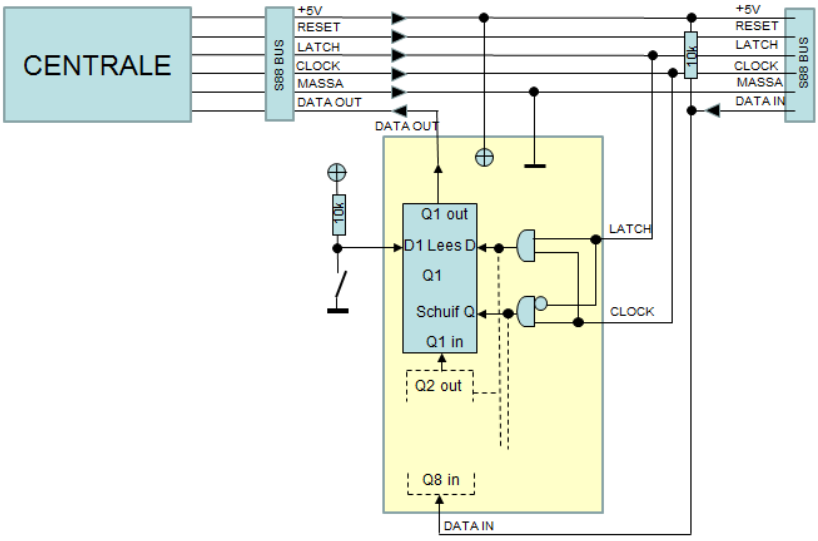
Er wordt in dit onderzoek met twee termen gewerkt waarmee hetzelfde bedoelt wordt, dat zijn LATCH en Load, in de onderstaande afbeelding wordt de term LATCH gebruikt maar in het verslag wordt de term Load gebruikt.

De werking is als volgt, op het moment dat het Xilinx bord een positieve flank geeft op de CLOCK en de Load hoog is, dan zal het schuifregister de data op Ingang D inlezen. Geeft het Xilinx een positieve flank en is de LATCH Laag, dan zal het schuifregister één positie opschuiven. En kan dus dus de volgende bit uitgelezen worden.
Hoe de S88 bus de parralelle input omzet naar seriele output is voor dit onderzoek niet van belang en valt buiten de scope van dit verslag.



Figuur 5.1: Tijdschema van het S88 protocol.

@onlineID,
title = S88 TERUGMELDERS,
date = 04-03-2016,



Figuur 5.2: Schematische tekening van het S88 protocol.

url = http://users.telenet.be/RedDeBist/MBAAN/S88%20terugmelder.htm

De rechter S88 bus zet paralelle data om naar seriele data, hoe dit gebeurt valt buiten de scope van dit onderzoek. Er wordt hier alleen onderzocht hoe een Xilinx bord met dit protocol kan communiceren.

5.1.2 Hoe wordt het XILINX Spartan 3E FPGA bord gebruikt?

Het FPGA bord krijgt instructies in de Hardware programeer taal VHDL om via het S88 protocol te controleren of er iets gebeurd op het RM88 bord. Dit houd in dat het FPGA bord een 'Clock' nodig heeft om te reageren op inkomende 'data' van het RM88 bord.
Dit wordt tot stand gebracht door op het FPGA bord via GPIO poorten verbinding te leggen naar het eerder vernoemde RM88 bord. To be continued

Hoofdstuk 6

Ontwerp

Nu alle deel vragen beantwoord zijn in de Analyse kan er verder gegaan worden met het ontwerpen van de werking van het XILINX bord.

6.1 Textueel ontwerp van het programma

De volgende elementen moeten ontworpen en gerealiseerd worden:

1. De clock naar beneden schalen zodat de Leds door mensen te zien zijn
2. De S88 Signalen moeten nagebootst worden

In dit hoofdstuk zal het ontwerpen plaatsvinden, dus in woorden uitgelegd wat er gedaan moet worden, en in het hoofdstuk realiseren wordt het in VHDL gerealiseerd.

6.2 Custom Clockrate genereren

Het eerste wat gedaan moet worden is de clockrate naar beneden schalen, dit wordt gedaan omdat de normale clockrate 50 MHz is.

Deze clockrate is te hoog om met een menselijk oog te kunnen waarnemen, hierom zal de clockrate intern verlaagt moeten worden.

Dit zal gebeuren door middel van een interne timer, die iedere clocktick met één verhoogd wordt.

Op het moment dat deze timer, die vanaf dit moment ClockTimer genoemt zal worden, een waarde van 2.5×10^4 bereikt zal er een andere variabele genaamt CustomClock met getoggled worden.

Dit houdt in dat er een eigen clock wordt gegenereerd die 0.5 seconde hoog is om daarna 0.5 seconde laag te zijn. Dit betekend dat de gegeneerde clock ofwel CustomClock een clockrate van 1 Hz heeft.
Dit is goed zichtbaar voor het menselijke oog.

6.3 De S88 Signalen

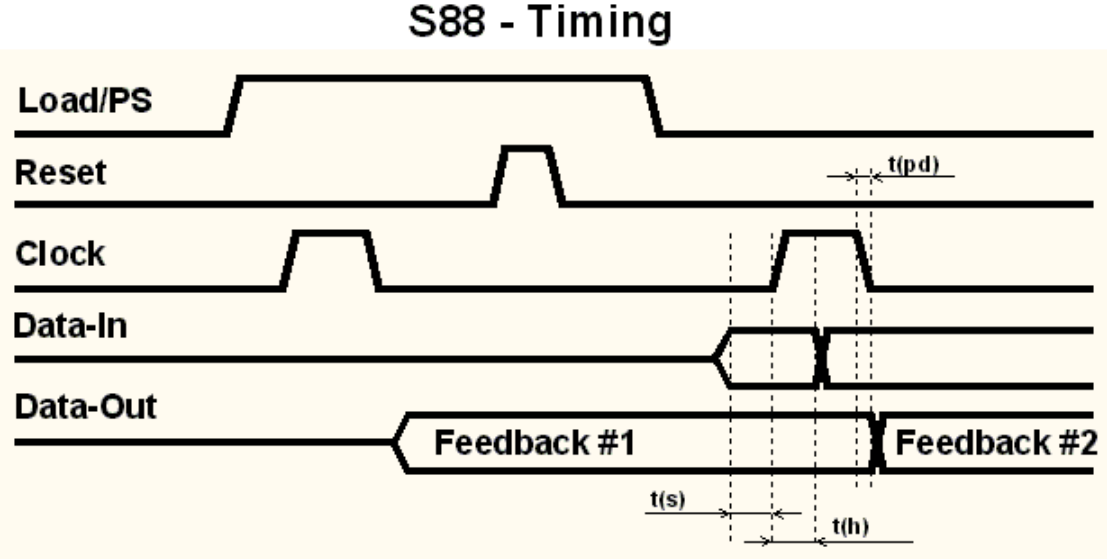
In de Analyze in de sectie ”Hoe werkt een s88 protocolïs reeds uitgelegd hoe het s88 protocolr werkt.
Hier is uit af te leiden dat er door er gebruikt gemaakt wordt van de volgende IO poorten:

- CLOCK
- DATA-OUT
- LOAD
- RESET

Het ontwerpen van deze signalen wordt opgesplitst in twee secties, De initialisatie en het daadwerkelijke uitlezen van het schuifregister.

6.4 Initialisatie van het schuifregister

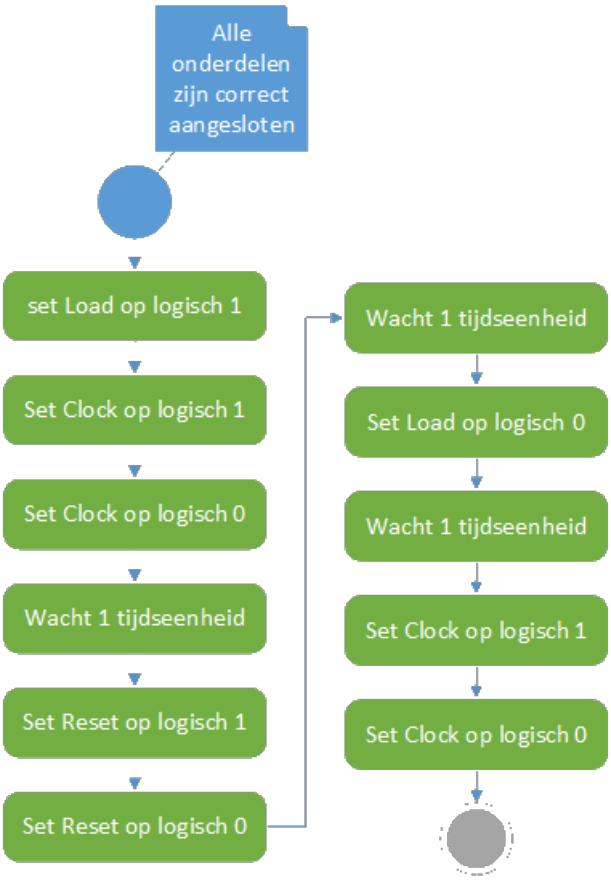
De volgorde van deze signales wordt beschreven in onderstaande image.



Deze image geeft aan welke poorten wanneer op een logisch 1 gezet moeten worden. Als voorbeeld, de lijn onder Load/PS is de lijn die bij

Load hoort. In het begin geeft de lijn een logisch 0 weer, de stijding betekent dat daar de poort op een logisch 1 gezet wordt.

Van de poorten Load/PS, Reset en Clock is een UML activiteiten diagram gemaakt, dit is gedaan zodat de volgorde van de signalen overzichtelijker weer te geven is.



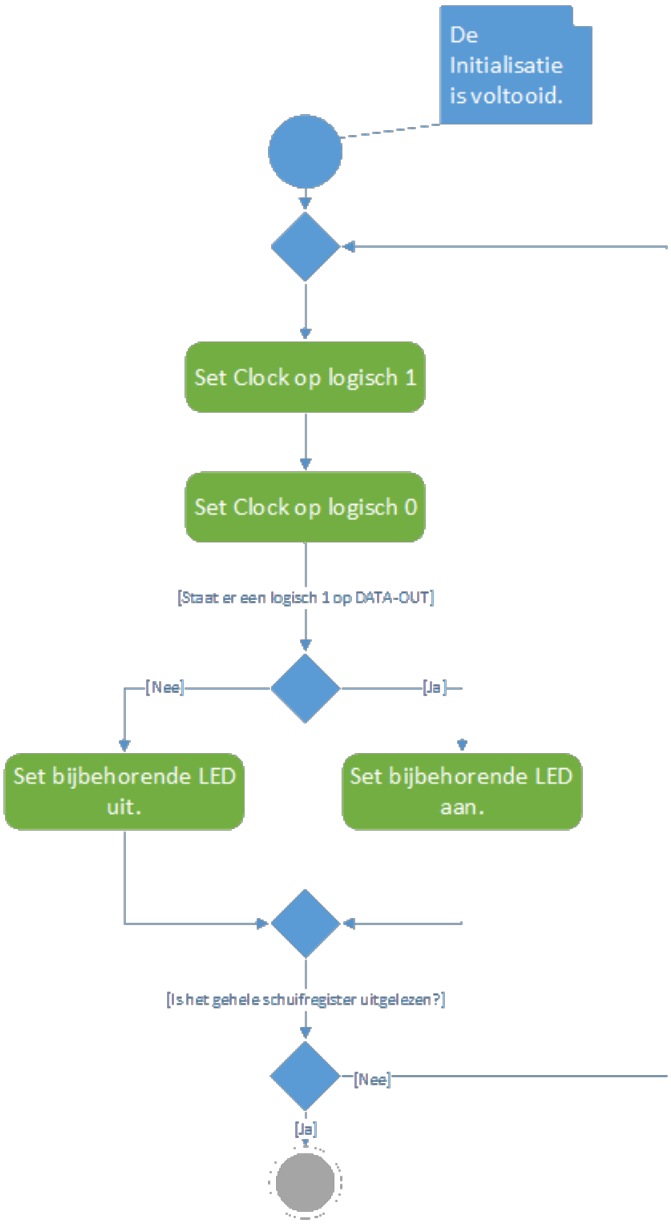
Hierin staat één tijdseenheid voor één seconde en is de duur van elke opdracht gelijk aan één tijdseenheid, dus één seconde.
Het moet gelezen worden door middel van de pijlen waarin er bovenaan begonnen met lezen wordt bij de blauwe cirkel.
Er mag alleen begonnen aan de opdracht als "Alle onderdelen zijn correct aangesloten"waar is, anders heeft het uitvoeren van de in de diagram beschreven opdrachten geen zin.

Als er dus gecontroleerd is dat alles correct aangesloten is kan er begonnen worden met de eerste opdracht, in dit geval ”Set Load op logisch 1”. Nadat deze opdracht uitgevoerd is kan er verdergegaan worden met de opdracht die door middel van de pijl verbonden is, in dit geval ”Set Clock op logisch 1”. Zo wordt er verdergegaan totdat alle opdracht uitgevoerd zijn, als alle opdrachten uitgevoerd zijn is het schuifregister succesvol geïnitialiseerd.

6.5 Uitlezen van het Schuifregister

Nadat de initialisatie voltooid is kan er begonnen worden met het uitlezen van het schuifregister. Zoals is beschreven in de analyse kan het schuifregister op de volgende manier uitgelezen worden, er wordt één tijdseenheid een logische 1 op Clock gezet, op de falling-edge van de Clock komt de volgende bit van het schuifregsiter op Data-Out gezet.

Dus nadat de Data-Out uitgelezen te hebben kan door middel van de Clock één tijdseenheid een logisch 1 te maken het volgende bit uit het schuifregister op de Data-Out gezet worden. En op die manier kan het hele schuifregister uitgelezen worden.



Wat hier niet goed weergegeven is is dat het net zo vaak uitgevoert zal worden totdat alle bits uit het schuifregister uitgelezen zijn, normaal gesproken wordt dit gedaan met een Loop maar omdat er bij de vereisten al vastgelegd is dat er gebruik gemaakt zal worden van de hardware beschrijving taal VHDL zal dit anders verlopen.

VHDL beschikt namelijk wel degelijk over een loop maar deze zal hier niet gebruikt worden, de redenatie hierachter is als volgt. Alle elementen worden geclockt op de CustomClock. Een loop kan niet geclockt worden en zal dus zal niet goed kunnen omgaan met de rest van het systeem.

Wat er wel gedaan zal worden is het volgende, vanaf het moment dat de initialisatie voltooid is zal er om de tijdseenheid de Clock voor één tijdseenheid logisch 1 gemaakt worden. De tijdseenheid die er dan tussen zit zal gebruikt worden om Data-Out uit te lezen. Dit zal net zo lang doorgaan totdat alle bits uit het schuifregister door het systeem uitgelezen zijn.

Hoofdstuk 7

Implementatie

Voordat er begonnen kan worden wordt er eerst een korte uitleg gegeven over de structuur, syntax en code van de hardware beschrijving taal VHDL.

7.1 Hoe wordt VHDL gebruikt

VHDL begint met een declaratie van alle gebruikte libraries, dit zijn bibliotheken waarin staat hoe de VHDL code met het XILINX bord om moet gaan, bijvoorbeeld waar de aansluitingen zitten.

Na het declareren van de libraries moeten alle IO poorten van het systeem gedeclareerd worden.

Na het declareren van de libraries en de IO poorten kan er nog gebruikt gemaakt worden van interne variabelen, signals genoemt. Waar de IO poorten alleen binair 0 of 1 kunnen zijn kunnen signals meer informatie bevatten, het is mogelijk om integers, bools en dergelijke te gebruiken.

Nadat alles gedeclareerd is kan er begonnen worden met de daadwerkelijke functies.

7.2 Declaratie VHDL

De volgende libraries moeten gedeclareerd worden om de VHDL te kunnen communiceren met het XILINX bord.

- ieee;
- ieee.std logic 1164.all;
- ieee.std logic unsigned.all;
- ieee.numeric std.all;

7.3 Initialisatie van het schuifregister

Hieronder wordt een lijst gegeven van alle ***uitgangen*** die gebruikt gaan worden:

- LED0 t/m LED7.
  Deze LEDs worden gebruikt om de toestand van de uit Data-Out ingelezen bits te weer te geven, ofwel 1(Led aan) ofwel 0(led uit).
- GPIO16
  Dit is de IO poort waarop Load aangesloten wordt.
- GPIO17
  Dit is de IO poort waarop Reset aangesloten wordt.
- GPIO14
  Dit is de IO poort waarop Clock aangesloten wordt.

In de code zal de naam die ook gebruikt wordt in het tijdschema gebruikt worden, hierdoor is de code beter leesbaar. Er staat bijveerbeeld Load in plaats van GPIO16.

Hieronder wordt een lijst gegeven van alle ***Ingangen*** die gebruikt gaan worden:

- GPIO12
  Dit is de IO poort waarop Data-Out aangesloten wordt.
- SW0
  Dit is de IO poort die aangeeft of de upper of lower byte gelezen wordt.
- Onboardclock
  Dit is de eerdergenoemde interne clock, deze is geklokt op 50 MHz.

In de code zal de naam die ook gebruikt wordt in het tijdschema gebruikt worden, hierdoor is de code beter leesbaar. Er staat bijveerbeeld DataOut in plaats van GPIO12.

7.4 Uitlezen van het Schuifregister

Omdat er in totaal twee bytes tegelijk ingelezen kunnen worden met behulp van het schuifregister wordt er een Switch gebruikt om aan te geven welke Byte er uitgelezen moet worden. Hier wordt constant op gecontroleerd bij het uitlezen van de gegevens, deze Zwitch is al genoemd bij de lijst met ingangen.

Het uitlezen van de registers is een constante herhaling van bijna dezelfde handelingen, deze handelingen bestaan uit het volgende:

1. Zet een logische 1 op de Clock.
2. Zet een logische 0 op de Clock.
3. Controleer bij de eerste byte of de Switch een logisch 0 is en bij de tweede byte of de Switch logisch 1 is.
4. Is dit waar dan kan de bijbehorende LED gelijk gesteld worden aan DataOut. Zo niet dan gebeurt er niets.

Hierbij vind nummer één zich plaats op tijd is x en nummer 2 tot en met 4 vinden zich plaats op tijd is x + 1.

Er is voor gekozen om dit in een Switch Case opstelling te plaatsen, hierbij wordt het bitpatroon van een variable vergeleken om te achterhalen welke opdracht er uitgevoert moet worden. Dit levert een code op die korter is dan met het gebruik van If statements.

Omdat dit een constante herhaling is van bijna dezelfde handelingen

Hoofdstuk 8

Apendix

```
1  library IEEE;
2  use IEEE . STD_LOGIC_1164 .ALL;
3  use IEEE . NUMERIC_STD .ALL;
4
5
6  --library UNISIM;
7  --use UNISIM.VComponents.all;
8  entity s88 is
9      port
10     (
11         OnboardClock, DataOut, HighBit : in
               std_logic;
12         LED0 , LED1 , LED2 , LED3 , LED4 , LED5 ,
               LED6 , LED7 , Load, Reset, Clock : out
               std_logic
13     );
14 end s88;
15
16 architecture s88Timing of s88 is
17     signal TimingCounter : unsigned (24 downto 0) :=
           ( others => '0');
18     begin
19         --Genereren van de CustomClock
20         timer : process(OnboardClock)
```

```
21          --Variabelen
22          variable ClockCounter : unsigned (24 downto
               0) := (others => '0');
23          variable TijdseenheidCounter : integer := 0;
24          --daadwerkelijk process
25          begin
26              if(rising_edge(OnboardClock)) then
27                  ClockCounter := ClockCounter + 1;
28                  TimingCounter <= ClockCounter;
29                  if(
30                  TimingCounter(0) = '0' and
        TimingCounter(1) = '0' and TimingCounter(2) = '0'
         and TimingCounter(3) = '0' and TimingCounter(4)
        = '0' and TimingCounter(5) = '0' and
        TimingCounter(6) = '1' and TimingCounter(7) = '0'
         and TimingCounter(8) = '0' and TimingCounter(9)
        = '0' and TimingCounter(10) = '0' and
        TimingCounter(11) = '1' and TimingCounter(12) =
        '1' and TimingCounter(13) = '1' and TimingCounter
        (14) = '1' and TimingCounter(15) = '0' and
        TimingCounter(16) = '1' and TimingCounter(17) =
        '0' and TimingCounter(18) = '1' and TimingCounter
        (19) = '1' and TimingCounter(20) = '1' and
        TimingCounter(21) = '1' and TimingCounter(22) =
        '1' and TimingCounter(23) = '0' and TimingCounter
        (24) = '1') then
31                      TimingCounter <=
                           "00000000000000000000000000";
32                      TijdseenheidCounter :=
                           TijdseenheidCounter + 1;
33                      --init
34                      case TijdseenheidCounter is
35                          when 0 =>
36                          -- niks
37                          when 1 =>
38                          Load <= '1';
39                          when 5 =>
40                          Reset <= '1';
```

```
41                          when 6 =>
42                          Reset <= '0';
43                          Load <= '1';
44                          when others =>
45                          --niks
46
47                      end case;
48                      --Leds
49                      case TijdseenheidCounter is
50  when 2 =>
51   Clock <= '1';
52   when 3 =>
53   Clock <= '0';
54   if(HighBit = '0') then
55   LED0 <= DataOut;
56   end if;
57   when 8 =>
58   Clock <= '1';
59   when 9 =>
60   Clock <= '0';
61   if(HighBit = '0') then
62   LED1 <= DataOut;
63   end if;
64   when 14 =>
65   Clock <= '1';
66   when 15 =>
67   Clock <= '0';
68   if(HighBit = '0') then
69   LED2 <= DataOut;
70   end if;
71   when 20 =>
72   Clock <= '1';
73   when 21 =>
74   Clock <= '0';
75   if(HighBit = '0') then
76   LED3 <= DataOut;
77   end if;
78   when 26 =>
```

```
79   Clock <= '1';
80   when 27 =>
81   Clock <= '0';
82   if(HighBit = '0') then
83   LED4 <= DataOut;
84   end if;
85   when 32 =>
86   Clock <= '1';
87   when 33 =>
88   Clock <= '0';
89   if(HighBit = '0') then
90   LED5 <= DataOut;
91   end if;
92   when 38 =>
93   Clock <= '1';
94   when 39 =>
95   Clock <= '0';
96   if(HighBit = '0') then
97   LED6 <= DataOut;
98   end if;
99   when 44 =>
100  Clock <= '1';
101  when 45 =>
102  Clock <= '0';
103  if(HighBit = '0') then
104  LED7 <= DataOut;
105  end if;
106  when 50 =>
107  Clock <= '1';
108  when 51 =>
109  Clock <= '0';
110  if(HighBit = '1') then
111  LED0 <= DataOut;
112  end if;
113  when 56 =>
114  Clock <= '1';
115  when 57 =>
116  Clock <= '0';
```

```
117   if(HighBit = '1') then
118   LED1 <= DataOut;
119   end if;
120   when 62 =>
121   Clock <= '1';
122   when 63 =>
123   Clock <= '0';
124   if(HighBit = '1') then
125   LED2 <= DataOut;
126   end if;
127   when 68 =>
128   Clock <= '1';
129   when 69 =>
130   Clock <= '0';
131   if(HighBit = '1') then
132   LED3 <= DataOut;
133   end if;
134   when 74 =>
135   Clock <= '1';
136   when 75 =>
137   Clock <= '0';
138   if(HighBit = '1') then
139   LED4 <= DataOut;
140   end if;
141   when 80 =>
142   Clock <= '1';
143   when 81 =>
144   Clock <= '0';
145   if(HighBit = '1') then
146   LED5 <= DataOut;
147   end if;
148   when 86 =>
149   Clock <= '1';
150   when 87 =>
151   Clock <= '0';
152   if(HighBit = '1') then
153   LED6 <= DataOut;
154   end if;
```

```
155   when 92 =>
156   Clock <= '1';
157   when 93 =>
158   Clock <= '0';
159   if(HighBit = '1') then
160   LED7 <= DataOut;
161   end if;
162
163   when others =>
164   -- niks
165                       end case;
166
167                   end if;
168               end if;
169   end process;
170   end s88Timing;
```